Escuela Técnica Superior de
Ingeniería Informática

Deliverable-04

Lint Report Workgroup: E7.04 Date: 23/05/2022

Miembros:

Baños González, Alejandro (alebangon@alum.us.es)

Flores Rodríguez, Román (romflorod@alum.us.es)

Grosso Gómez de Terreros, Javier (javgrogom@alum.us.es)

Gutiérrez Ceballos, Pablo (pabgutceb@alum.us.es)

Ibáñez Montero, Julia (julibamon@alum.us.es)

Roldán Cadena, Jesús (jesrolcad@alum.us.es)

Repositorio de Github: https://github.com/jesrolcad/Acme-Toolkits

Índice

1. Resumen ejecutivo

Este documento detalla los resultados obtenidos tras el análisis del código con la herramienta SonarLint. Para ello, se muestran capturas de los resultados y se especifica la solución que hemos proporcionado para eliminar por completo los malos olores de nuestro código.

Hemos observado que los malos olores se han concentrado en los métodos validate de InventorToolkitPublishService y en el InventorToolkitListService. En concreto, hemos cometido fallos al hacer uso de los tipo booleanos, ya que hemos igualado a false o true algunos de ellos (que toman uno de esos valores de por sí). Además, hemos incluido algunas líneas de código innecesarias ya que, en algunas ocasiones, no es necesario crear variables para llamar a un método del repositorio cuando podemos llamar a este método directamente desde el return y que nos devuelva ahí directamente lo que queremos obtener (sólo cuando no vamos a modificar nada de lo que nos proporciona el repositorio).

Hemos concluido que el trabajo realizado para deshacernos de los malos olores no nos ha supuesto una gran carga de trabajo ya que hemos comprendido el porqué de estos malos olores y hemos sabido solucionarlos sin ningún problema. Además, hemos obtenido un número muy bajo de malos olores.

Hemos considerado, aunque en el anterior sprint decidimos lo contrario, volver a hacer un análisis del código una vez estuvieran todas las funcionalidades implementadas, ya que durante el desarrollo hemos ido modificando o eliminando en numerosas ocasiones los distintos métodos y que, si hubiésemos hecho un seguimiento regular de los malos olores del código, habríamos tenido que arreglar dichos bad smells de código que iba a ser eliminado o modificado de nuestro proyecto, por lo que nos habría supuesto una pérdida de tiempo y esfuerzo.

2. Tabla de control de cambios

Número de revisión	Descripción	Fecha
1	Creación del documento y sus apartados	22/05/2022
2	Redacción completa del documento	23/05/2022

3. Introducción

En este documento se reflejan los resultados obtenidos tras hacer un análisis del código con la herramienta SonarLint. Se van a mostrar capturas de los malos olores detectados por la herramienta, así como una explicación de los cambios que hemos realizado para solucionarlos. Para ello, hemos organizado este documento en distintas secciones descritas a continuación:

Sección 4. *Resultados obtenidos*. En esta sección se muestran capturas del análisis de nuestro código proporcionado por SonarLint.

Sección 5. *Cambios realizados.* Se explican los cambios realizados tras el análisis del código, así como una captura final para mostrar la solución a los malos olores.

Sección 6. *Conclusiones.* Se reflejan las conclusiones que hemos obtenido del análisis del código con SonarLint y se hace un balance de la metodología que hemos seguido para la detección y eliminación de malos olores.

Sección 7*. Referencias bibliográficas*. Esta sección está vacía ya que no hemos recurrido a ningún tipo de bibliografía para la redacción de este documento.

Finalmente, el documento posee la siguiente estructura: portada, índice, resumen ejecutivo, tabla de control de cambios, introducción, contenidos (divididos en las secciones *Resultados obtenidos y Cambios realizados)*, conclusiones y referencias bibliográficas.

4. Resultados obtenidos

Tras hacer el análisis de nuestro código (finalizado) con la herramienta SonarLint, hemos obtenido un total de 4 malos olores. Dichos malos olores se encuentran en los PublishService de InventorToolkit y en el InventorToolkitItemListService. Estos malos olores los agrupamos en dos.

Malos olores obtenidos:

Resource	Date	Description
InventorToolkitItemListService.java		Immediately return this expression instead of assigning it to the temporary variable "quantities".
InventorToolkitPublishService.java		Remove the unnecessary boolean literal.
InventorToolkitPublishService.java		Remove the unnecessary boolean literal.
InventorToolkitPublishService.java		Remove this expression which always evaluates to "true" [+3 locations]

El primer mal olor, en el ToolkitItemListService, nos indica que no es necesario crear la variable temporal quantities, sino que podemos devolverla directamente en el return.

```
@Override
public Collection<Quantity> findMany(final Request<Quantity> request) {
    final int toolkitId;
    toolkitId = request.getModel().getInteger("masterId");
    final Collection<Quantity> quantities = this.repository.findQuantityByToolkitId(toolkitId);

    return quantities;
}
```

En segundo lugar, en el InventorToolkitPublishService, SonarLint nos indica que en las líneas 121 y 128 del método validate tenemos código inútil. Este mal olor se debe al items!=null && items.isEmpty()==false y al publishItem==true.

```
121         errors.state(request, items!=null && items.isEmpty()==false, "*", "inventor.toolkit.form.error.no-items");
122
123         for (final Item item : items) {
124             publishItem= publishItem && item.isPublished();
125         }
126
127
128         errors.state(request, publishItem==true, "*", "inventor.toolkit.form.error.no-items-published");
129
```

5. Cambios realizados

Para paliar el primer mal olor, hemos cambiado el método findMany() del InventorToolkitListService, hemos eliminado la variable quantities y llamado al repositorio directamente desde el return.

```
@Override
public Collection<Quantity> findMany(final Request<Quantity> request) {
    final int toolkitId;
    toolkitId = request.getModel().getInteger("masterId");

    return this.repository.findQuantityByToolkitId(toolkitId);
}
```

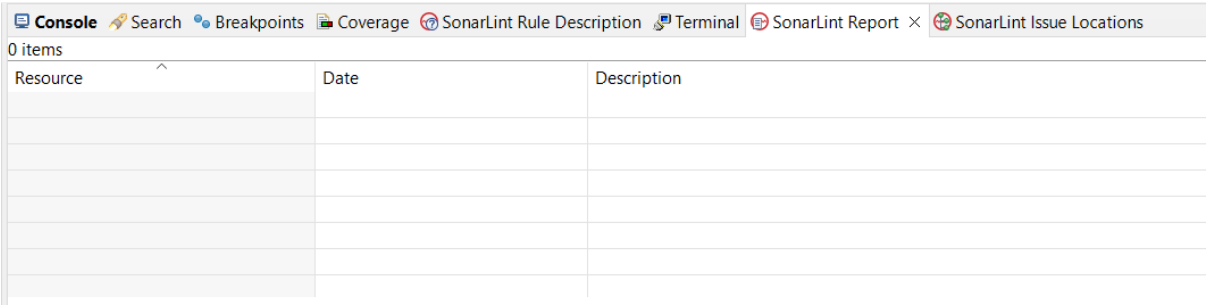
Para el segundo mal olor, en el método validate de InventorToolkitPublishService hemos eliminado los booleanos que no aportan nada reajustando el código de la siguiente manera:

```
for(final Quantity quantity: quantities) {
    final int id=quantity.getId();
    final Collection<Item> item=this.repository.findManyItemByQuantityId(id);
    items.addAll(item);
}
errors.state(request, !items.isEmpty(), "*", "inventor.toolkit.form.error.no-items");

for (final Item item : items) {
    publishItem= publishItem && item.isPublished();
}


errors.state(request, publishItem, "*", "inventor.toolkit.form.error.no-items-published");
```

Tras hacer estos cambios hemos conseguido eliminar todos los olores sin afectar en absoluto a la funcionalidad del código:



6. Conclusiones

Tras haber hecho el análisis del código con SonarLint hemos concluido que hemos bajado considerablemente el número de bad smells con respecto al anterior sprint. Estos bad smells se han concentrado en un mal uso de los tipo booleanos en los métodos validate y código redundante.
Gracias al análisis que nos ha proporcionado SonarLint hemos aprendido varias cosas:

En primer lugar, hemos aprendido la importancia que tiene no poner código redundante, es decir, no es necesario crear variables para obtener, por ejemplo en este caso, una colección cuando podemos devolverla directamente. Habría tenido sentido crear la colección quantities si hubiésemos hecho algún tratamiento sobre ella, pero al devolverla tal y como la proporcionaba el método del repositorio, es mejor devolverla directamente desde el return (llamando al repositorio desde aquí) para reducir el número de líneas de código.

También, hemos aprendido a hacer un buen uso de los tipo booleanos en las validaciones. Cuando queremos comprobar, por ejemplo, que nos salte un error cuando un booleano sea false, no es necesario igualarlo a false ya que no aporta nada en el código. Al ejecutar la validación el booleano ya va a tomar uno de los valores (false o true) por lo que no tiene ningún sentido igualarlo a uno de ellos.

Nuestra metodología para la detección de malos olores ha sido ejecutar SonarLint sobre una rama creada desde master en el momento en el que todas las funcionalidades ya habían sido implementadas. Aunque en el anterior sprint consideramos que para este ibamos a cambiar nuestra metodología (llevando un seguimiento periódico de los malos olores para no cometer los mismos fallos una y otra vez durante el desarrollo), hemos decidido que iba a ser una buena práctica hacerlo como en el anterior sprint (con todo el código terminado), ya que en este sprint hemos ido cambiando y eliminando muchos métodos cada semana, por lo que habría significado ir eliminando malos olores de métodos que podrían haber sido eliminados por completo posteriormente. Por tanto, hemos decidido volver a la anterior metodología, ya que así no hemos perdido tiempo arreglando bad smells de código que iba a pasar a no formar parte del proyecto de cara a la entrega.

En este caso, hemos obtenido tan pocos bad smells ya que las funcionalidades que hemos implementado están basadas en Acme Jobs y no hemos hecho uso de etiquetas (como en los dashboards anteriormente), que fue donde detectamos la inmensa mayoría de malos olores en el anterior sprint.

7. Referencias bibliográficas